Third-Party Flow Control*

Dushyant Bansal and Paul A.S. Ward
Shoshin Research Group
Department of Electrical and Computer Engineering
University of Waterloo
{dbansal,pasward}@shoshin.uwaterloo.ca

Abstract

Flow control is critical to the efficient operation of Internet Service Providers network equipment. In particular, the ability to effectively shape traffic can reduce cost and improve overall customer satisfaction. While such traffic shaping is typically performed by an inline traffic shaper, there are a number of practical cases in which such an inline approach is not feasible. In particular, an inline traffic shaper may reduce reliability or simply be against ISP policy. In these cases third-party flow control is required.

Third-party flow control allows the shaper to see all traffic and to inject new traffic into the network. However, it does not allow the shaper to remove or modify existing network data. Within these limitations we study two techniques for flow control, triple-ACK duplication and zero-window-size acknowledgement. We provide analytical justification for why these techniques are promising. In addition, we demonstrate, via simulation, that the zero-window-size technique can reduce bandwidth consumption by 40%, while the triple-ACK duplication can reduce it by up to 85%. These techniques thus offer the possibility for significant flow-control capabilities by a third-party traffic shaper.

Keywords: TCP, flow control, congestion control, third-party, peer-to-peer.

1. Motivation

Excessive packets sent in a network create long packet queues at routers, which may lead to buffer overflows, resulting in packet drops. This results in retransmissions, which consequently reduce the goodput. Further, Internet Service Providers (ISPs) are typically billed according to their peak throughput, and thus desire to limit it, while still providing good service to their clientele. ISPs which manage their traffic typically install a traffic shaper inline with their router. This shaper may form part of newer routers. This mode of deployment is shown in Figure 1.

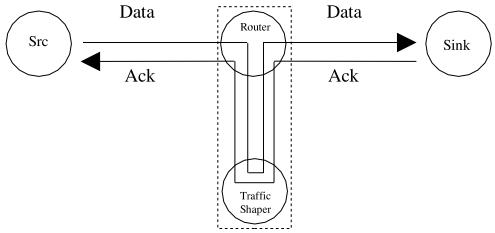

Figure 1. Inline Traffic-Shaper Deployment

However, in some cases, it is not desirable to install an external vendor's traffic shaping box inline with the router for fear of losing all connectivity if that box fails. An alternative is to install the traffic shaper in parallel to the router equipment, as shown in Figure 2. A traffic shaper in this scenario gets copies of all packets and is able to inject packets into the traffic stream but is not able to control the flow of existing packets. Such a third-party deployment mitigates the problem described above by decoupling router failure and operation from the reliability of the traffic shaper.

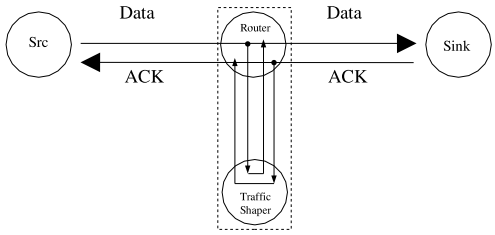

Figure 2. Third-Party Shaper Deployment

*This work was supported in part by Sandvine Inc. and NSERC

In this paper we design and evaluate techniques for third-party traffic shaping. Since most Internet traffic runs over the Transmission Control Protocol (TCP), our work deals only with solutions based on TCP traffic. This work was motivated by file-sharing over peer-to-peer (P2P) networks, which is the greatest consumer of Internet bandwidth today, and thus we assume that the TCP flows we wish to control are long lived.

The remainder of this paper is organized as follows. In Section 2 we describe the basics of TCP, and existing techniques for flow control. We then develop two novel approaches for third-party TCP flow control based on replicating acknowledgement packets, either with zero-window size, to force flow control, or threefold acknowledgement duplication to force congestion control. In Section 4 we evaluate the efficacy of our approaches using the ns-2 simulator [13], demonstrating that the techniques can reduce bandwidth consumption of a TCP flow by up to 85%. We conclude by describing what work remains for this technique to be applicable for deployment.

2. Background

We now describe the operation of TCP and discuss existing methods devised to control TCP traffic.

2.1. Basic mechanics

TCP is a byte-stream-based connection-oriented protocol that uses sliding windows to provide reliability, in-order delivery of segments to the receiving application, flow control, and congestion control. A sliding window is essentially a window of data segments that are to be transmitted or received. Given that a sender requires an acknowledgement (hereafter, ACK) for every segment it transmits, the sender can only have as many segments unacknowledged as the sender window allows. It stops sending segments if no segment in its current window has been acknowledged yet. As ACKs arrive, the sliding window moves forward, as shown in Figure 3. This provides for reliable communication.

Given that a receiver acknowledges all segments it receives (cumulatively or individually), the receiver can only accept an additional receiver-window worth of segments beyond the last segment it has acknowledged. Anything outside this window is dropped. As the receiver application consumes the bytes, the receiver window becomes freed. This is shown in Figure 3. To prevent wasteful transmissions, the TCP receiver advertises the amount of space it has available to the sender as part of the header of every ACK. This informs the sender how many more bytes the receiver can accept at that moment. The sender will never send more data than the receiver's advertised window, thus implementing flow control. It may, however, send less, according to the sender's understanding of the congestion level in the network.

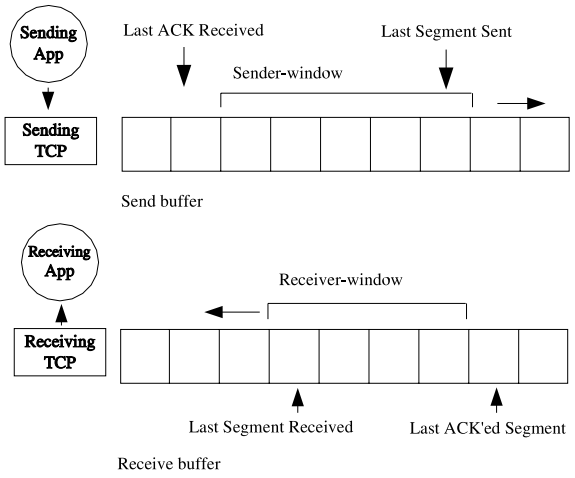

Figure 3. TCP sliding window protocol [12]

2.1.1 Congestion window

The basic model of TCP is called the Tahoe version. The following is a simplification of how TCP Tahoe operates. TCP Tahoe initializes its sliding window at 2 segments and increments the window size by 1 for every ACK received, until the window size reaches the congestion threshold. This mode of operation is called slow-start. Thereafter, the window size increases by 1 segment after an entire window of data is acknowledged.

To detect congestion in the network, the sender uses a timeout mechanism to wait for acknowledgements from the receiver. If the sender times out waiting for an ACK for a packet, it infers that the packet is lost and assumes that it is because the network is congested. The sender retransmits the packet, halves its congestion threshold, and resets the congestion window to its initial setting, which would be 2 segments. The sender starts again in slow-start mode.

2.1.2 Fast retransmit and recovery

Not all packet loss is caused by congestion. Some packet loss is simply a random occurrence within the network. However, the time-out mechanism requires that the sender has to wait for the entire time-out period before it realizes that a packet has been lost, and it assumes the cause of loss is congestion. A modification adopted to circumvent this problem was to use fast retransmit, and it has been implemented in TCP Reno 1. In this mechanism, if the receiver receives a data packet out of order, it sends out an ACK for

the last data packet received in order, effectively duplicating the ACK for that last in-order packet. When three duplicate ACKs are received the sender infers that the data packet following that which has been acknowledged four times has been lost in the network due to congestion. It immediately retransmits that packet, sets its congestion threshold to half the current congestion window, and resets the congestion window back to 2.

The assumption of congestion is unnecessarily pessimistic. The data packet that was acknowledged 4 times in a row was not necessarily lost due to network congestion but due to physical loss. This optimistic assumption led the next version of TCP, Reno 2, to skip slow-start, and only set the congestion window to half its current value rather than resetting it all the way to 2. This approach is called fast-recovery. Reno 2 is the *defacto* standard TCP implementation in use today.

2.2. Existing techniques for flow control

Most existing techniques for TCP flow control require inline traffic control or modification of the TCP installed on the end nodes. As such, these methods are not directly applicable in third-party deployment since the third-party shaper has no ability to change packets. However, we describe these techniques, partly for completeness, but mostly to detail what parameters must be controlled by the shaper, and how they might be controlled.

2.2.1 In-Network flow control

An inline traffic-shaper has control over all the packets passing through that router. It can modify the contents, including the header, of the packets, drop packets, or delay them in order to reduce the bandwidth. The sending rate of any TCP sender is controlled by the following five things: availability of data, the congestion window size, the receiver window size, the round trip time (RTT), and the rate of acknowledgements.

A sending TCP entity can only send packets if there is application data available. While it is possible to perform actions to affect the availability of application data, it is inherently application specific. As such, for each new application, a new shaper technique must be developed, and is thus costly to implement and deploy. This approach has been used to control peer-to-peer traffic flow rates.

The congestion-window size of a sender TCP, if smaller than the receiver-window size, can limit the rate of TCP traffic flows. The Random Early Detection (RED) technique [7] manipulates this window size by preemptively discarding selected TCP packets, thus causing the congestion window to be reduced. Our triple-ACK duplication mechanism, described in Section 3.1, is based on this approach.

Similarly, the receiver window size, if smaller than the congestion-window size, can limit the rate of TCP traffic flows. Several techniques have therefore been developed based on artificially reducing the size of the receiver window, below that which is advertised by the receiver [3, 4, 8]. Our zero-window-size technique, which we describe in Section 3.2 is inspired by these approaches.

It is well-understood that TCP is proportionally fair to the inverse of the round trip time (RTT). As such, the longer the RTT, the lower the bandwidth for the stream. The RTT can be increased if the traffic shaper introduces a delay between subsequent ACKs, thus slowing down the rate at which the sender TCP clocks its output data, thereby reducing bandwidth. Similarly, the faster the rate of ACKs, the faster the sender TCP's congestion window will expand, resulting in a larger data output rate overall. We are not aware of existing shapers using these approaches.

There are a number of other inline flow-control techniques that are independent of the transport protocol used, including token and leaky buckets [14] and maintaining some unused bandwidth on a link at all times [1] to provide a buffer for bursty traffic. None of these approaches seem to provide insight for the design of third-party shapers.

2.2.2 End-node flow control

Rather than introduce in-network elements to shape current traffic, the protocol can be updated on the end-hosts. An updated TCP stack allows one side to inform the other of congestion, for one side to infer impending congestion from the current network dynamics, or for the receiver to selectively acknowledge which data packets it has received to avoid unnecessary retransmissions and to speed up transmissions of the necessary packets [6]. While there is various research in this area (*e.g.*, [9, 10]), it has deployment problems, and does not appear useful for current ISP traffic control.

2.2.3 Mixed method flow control

A mixed approach to bandwidth management is one that combines the above two techniques. Some algorithms are implemented on the end-node stacks and an in-network element is also used. The two would work together to control network utilization. A typical example in this area is the Explicit Congestion Notification (ECN) scheme [5], a modification of RED that avoids unnecessary packet drops. This scheme sets an ECN bit in the packet header. The transmitter is expected to respond to these ECN bits in the same way as fast recovery *sans* the packet retransmission. Other techniques include Random Early Marking [2]. The drawback with these schemes is that they suffer the problems of end-node flow control, needing updated protocol stacks are the end hosts. As such, we cannot rely on their presence for third-party flow control.

3. Third-Party Flow-Control Mechanisms

We have designed two mechanisms for third-party flow-control. The first, triple-ACK duplication, is based on the indirect manipulation of the sender's congestion window threshold, and is inspired by the RED mechanism. The second technique, zero-window-size acknowledgement, manipulates the sender's view of the receiver window size. We now describe these methods in detail, and show how they are distinct from existing techniques.

3.1. Triple-ACK duplication

Our triple-ACK duplication algorithm attempts to manipulate the congestion window and threshold of the sending TCP entity. To do so, it requires that the sender TCP agent is executing the fast retransmit protocol. The manipulation is caused by sending out three duplicate ACKs for some of the ACKs it has seen. When the TCP sender receives four ACKs with the same sequence and acknowledgement numbers, it will invoke congestion control. Depending on the particular variant, it will either enter slow start (Reno 1) or fast recovery (Reno 2). In either case, the congestion window and threshold are reduced. In addition, the sender will retransmit the data segment that it understands to be lost.

The reduction in the sizes of the congestion window and threshold should reduce the rate at which the sender transmits data. However, we must note that the retransmissions caused by invocation of fast retransmit will also mitigate that to some degree, as well as reducing the goodput of the flow. In particular, we have found that the use of triple-ACK duplication for every ACK packet seen results in an increase in total bandwidth consumption, as well as a complete collapse of goodput. Careful optimization of the frequency of triple-ACK duplication is critical to the correct operation of this method. We note further that the bandwidth consumed by ACKs going from the router to the sender will also increase.

The triple-ACK duplication technique is illustrated in Figure 4.

This triple-ACK duplication algorithm is similar to the RED technique, in that it forces a packet retransmission, and causes the congestion window and threshold to be reduced. However, it differs in that the packet retransmitted has, in fact, already been seen by the receiver. As such, there is only brief interval during which the three duplicate ACKs can be transmitted to the sender. If they do not arrive before the next ACK from the receiver, the sending TCP entity will likely ignore them. In the case of the RED technique, the packet is genuinely dropped, and the remainder of the RED technique is simply the normal TCP reaction to the loss of a packet.

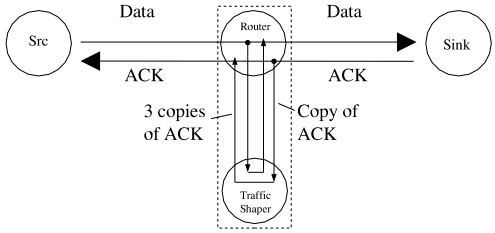

Figure 4. Triple-ACK duplication algorithm

Finally, we note that if the sender is using TCP Tahoe the triple-ACK duplication method will not work, as the data packet was not, in fact, lost, and thus TCP Tahoe will not enter congestion control. Again, this is in contrast to the RED technique, which will work regardless of TCP algorithm. We must emphasize, however, that the RED technique is not feasible in the third-party shaper deployment, as it requires the removal of a packet from the packet stream, which is strictly not possible for a third party.

3.2. Zero-window size algorithm

Our second technique attempts to manipulate the receiver-window size as seen by the sender TCP. It operates by sending out a zero receiver-window-size duplicate ACK for every ACK seen. This is shown in Figure 5.

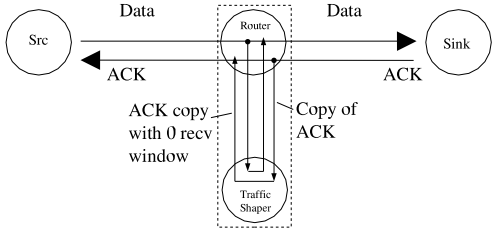

Figure 5. Zero-window-size algorithm

We note that while this technique is similar to inline algorithms that control the advertised receiver window size, it differs in that it can, at best, cause a brief reduction. Inline algorithms have strict control over the advertised window size, and thus whatever they set the advertised window size to, it will remain. In our third-party case, however, there is a stream of ACKs from the receiver, each of which will advertise the correct receiver window size. We can, thus, at best, briefly persuade the sender that there is no room at the receiver, before it finds out otherwise. It is for this reason that we maximize the reduction, by setting the advertised size to

0, which, again, differs from the approach in inline control, where such an action would not slow down the sender, but simply stop it.

4. Experimental Evaluation

To evaluate our techniques we developed a series of simulation experiments. Using the ns-2 simulator [13], we implemented the third-party flow-control architecture of Figure 2. Both the source and sink nodes have FullTCP agents attached to them. In ns-2 the FullTCP agent differs from the regular TCP agent in that it has support for connection setup and tear-down, as well as header flags. The header flags were necessary so that we could recognize ACK packets. The source node also had an application generating a steady stream of data using the "File Transfer Protocol" (FTP) attached to the TCP agent. Rather than implementing the FTP protocol, the application simply sent out a continuous stream of data. This is in keeping with the original motivation of the project, which is to manage large TCP flows, as exemplified by peer-to-peer file-sharing traffic.

The router node does not have a TCP agent attached. A traffic-shaper is connected to the router in third-party mode. For the purpose of the simulation, the router and traffic shaper are modeled as a single node. The communication delay between the router and traffic shaper is modeled simply as a time-delay. The router-cum-traffic-shaper node operates only at the link layer with no transport-layer functionality. Only the source and sink run TCP.

Unless otherwise specified, default ns-2 parameters were used. It is our intention to explore more variations in the future, to determine to what degree, if any, these defaults affect the accuracy of our results.

Given this setup, we performed a series of experiments using our two techniques. Our primary concern in these experiments was to determine the degree to which the methods could be used to control bandwidth, while operating as a third party. We now describe these results, starting with the triple-ACK duplication method.

4.1. Triple-ACK duplication results

The triple-ACK duplication method has one main control parameter, which is the frequency of duplication. That is, the technique can choose how many ACKs it sees before it generates a triple-ACK duplicate. Intuitively, if it is too frequent, the bandwidth reduction will be offset by excessive retransmission. Conversely, if it is too infrequent, the congestion window and threshold will not be adequately reduced.

We therefore performed a series of experiments in which the ACK-duplication rate was varied from 1, meaning every ACK from triple-duplicated, to 200, meaning that only one in 200 ACKs was duplicated. Duplication rates were increase by 1, for the first 10, then by 5, up to 25, then by 25, up to 100, and then by 50, up to 200. These experiments were performed over a range of bandwidths and delays, ranging from 1 Mbps, and 1 ms round trip time to 99 Mbps and 400 ms round trip time. Bandwidth was increased in increments of 2 Mbps. Round trip time was increased in increments of 1 ms for the first 10 ms, and every 10 ms thereafter. All combinations of these parameters were simulated, and full raw-result data is available at our website: http://www.ccng.uwaterloo.ca/˜pasward/ThirdParty/.

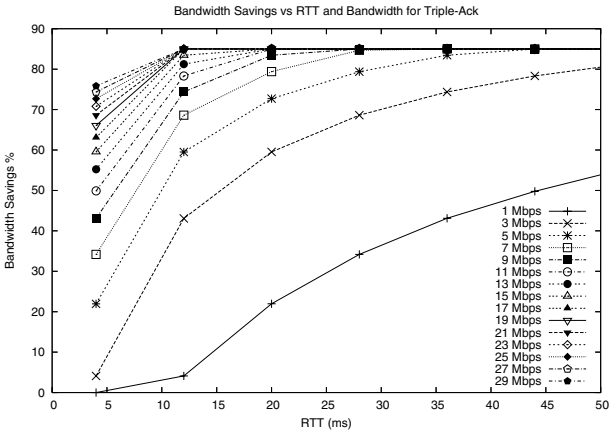

Figure 6. Sample triple-ACK duplication

The first, and unsurprising result, was that triple-ACK duplication of every ACK packet resulted in a bandwidth increase. Of some surprise, however, was the fact that triple-ACK duplication of every second ACK packet resulted in a bandwidth reduction though, not surprising;ly, the goodput reduction was rather severe. Of more significance, we found that in all instances, the bandwidth reduction reached a cap level, which was consistent regardless of bandwidth. The only factor that varied was that the round trip time determined the point at which that cap was reached.

Figure 6 shows a sample triple-ACK duplication, where the frequency of duplication was 6. This choice was dictated by the fact that this was the case of maximum bandwidth reduction we experienced. The graph shows only round trip times up to 50 ms, and bandwidths up to 29 Mbps. The reader should be able to see that further increase in the bandwidth simply produce reductions at the top of the graph. Likewise, when the round trip time is larger, the bandwidth reduction increases. It caps, in this case, at approximately 85% reduction. We note that the result of the retransmission in this case meant that the goodput dropped to approximately 83% of the total data transmitted. The goodput only exceeded 90% of total data transmitted when the frequency of triple-ACK duplication was less than

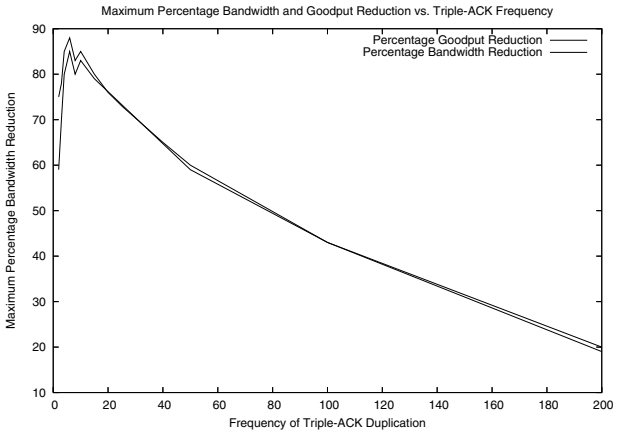

Figure 7. Maximum bandwidth reduction

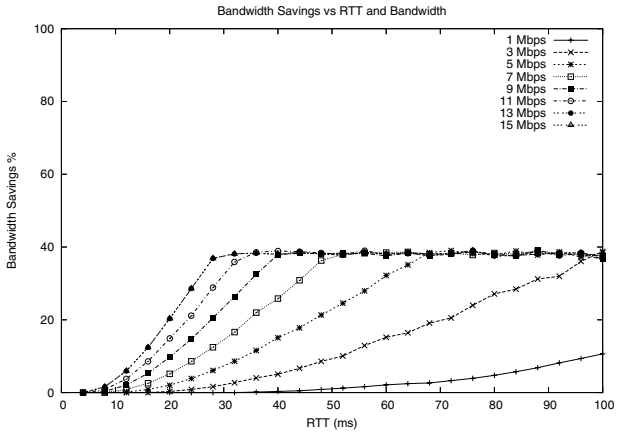

Figure 8. Bandwidth Savings vs Latency vs Bandwidth with the Zero-Window Algorithm

one in ten. However, it should be noted that at that point, the bandwidth reduction reached as high as 83%.

Figure 7 shows the maximum reduction achieved as a function of triple-ACK-duplication frequency. As can readily be seen, the maximum peaks in the mid-single digits, declining gracefully thereafter. What this figure does not show, however, is the reduction level at a given delay-bandwidth product. In particular, as is apparent in Figure 6, when the delay-bandwidth product is small, the reduction is noticeably less. Indeed, this problem gets worse, as the frequency of triple-ACK duplications declines. For example, when the bandwidth is 1 Mbps, and the round trip time 50 ms, the bandwidth reduction is approximately 53% at a duplication frequency of 6, while it is half that when the duplication frequency is 20.

The critical parameter range, representing typical home-based ISP customers using peer-to-peer file-sharing systems, is between zero to 30 ms round trip time, and mid- to low single digits bandwidth. In this range we can see from Figure 6 that, provided the delay is greater than about 20 ms, the triple-ACK-duplication scheme can reduce the bandwidth consumption by more than 20%. When the frequency is increased to 4, a somewhat lower round trip time will also experience bandwidth reduction.

These results suggest that triple-ACK duplication can be used effectively by a third-party-flow control system.

4.2. Zero-window-size results

The zero-window-size method has two main control parameters, which are the frequency of zero-window-size ACK generation and the "size" of the window advertised (which, in spite of our choice of name for this technique, need not be zero). However, for this series of experiments we simple set the advertised window size to zero, and generated a zero-window-size ACK for every true ACK the third-party shaper saw.

As with the triple-ACK duplication experiments, we varied the round trip time from 1 ms to 400 ms and the bandwidth from 1 Mbps to 99 Mbps. We used the same combinations of round trip time and bandwidth as in those prior experiments. Results for bandwidths between 1 Mbps and 15 Mbps and round trips times of between 1 ms and 100 ms are shown in Figure 8. As can be seen, the bandwidth reduction for this technique was capped at about 40%. The point at which this cap was reached was a function of the delay-bandwidth product. The larger the bandwidth, the quicker it reached the cap, though that point was capped at 13 Mbps. Close examination of the figure shows that the line for both 13 Mbps and 15 Mbps coincide. We note that there were no retransmissions with this algorithm. As a result, the goodput was equal to the throughput, and we did not need to calculate any goodput parameters *per se*.

Unfortunately, in the more-critical lower bandwidth-delay product area, this algorithm failed to provide significant bandwidth savings. For example, at 30 ms round trip time, with a link bandwidth of 5 Mbps, the reduction amounted to less than 10%. By contrast, the triple-ACK-duplication technique, at frequency 6, has a reduction of almost 70% at this delay-bandwidth point.

For our second set of experiments we varied the communication delay between router and traffic-shaper to observe the effect on bandwidth reduction. We did this not by adding a delay directly between the router and traffic-shaper but instead by adding a delay to when we send out the duplicate ACK with the modified receiver window size. This delay simulates the round trip time between router and traffic-shaper. We found that the effect was that the system either

operated as if there were no delay, or it failed to operate at all. That is, there was a hard switch over between the system functioning, and then ceasing to function. The delay permissible was found to equal the inter-ACK delay from the receiver. Thus, as long as the duplicate zero-window-size ACK was received before a new, true ACK was received, the system effectively reduced bandwidth consumption. This suggests that the system must be extremely responsive if it is to be effective.

5. Conclusion and Future Work

Based on our experiments, we have demonstrated that it is possible to control TCP flows with a third-party traffic shaper. Our triple-ACK-duplication algorithm was able to reduce bandwidth consumption by up to 85%, with a 15%r eduction in goodput. At lesser bandwidth-reduction levels, the goodput approaches 100%. The zero-window-size algorithm is also able to reduce bandwidth consumption, though by a lesser amount. It does not suffer from the need to retransmit, and thus maintains 100% goodput.

In the more-critical range of zero to 30 ms round trip time, and mid- to low single digits bandwidth, typical of home-based ISP customers using peer-to-peer file-sharing systems, the zero-window-size approach provides very little benefit. The triple-ACK duplication method is still able to provide more than 20% bandwidth reduction, with a duplication frequency of 6. At the expense of greater goodput reduction, duplication frequency can be increased, reducing the bandwidth at lower delay-bandwidth products.

Our current work is now attempting to address three points. First, we are looking at asymmetric latencies between the sender, receiver, and the third-party shaper. In particular, we believe that a typical deployment will have the shaper very close to either the sender or receiver, rather than being half-way between, as was simulated in our experiments. Second, we are implementing the triple-ACK duplication scheme, to determine the degree to which our simulation results work in practice. We are building this on top of the netfilter system [11]. Third, we are developing a controller for our system, to enable feedback-control-based bandwidth adjustment. The controller will measure the current bandwidth utilization, and compare it against the target level. The frequency of triple-ACK duplication will be adjusted accordingly.

References

[1] Yehuda Afek, Yishay Mansour, and Zvi Ostfeld. Phantom: A simple and effective flow control scheme. In *SIGCOMM '96: Conference proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 169–182. ACM Press, 1996.

[2] Sanjeewa Athuraliya, Steven H. Low, and David E. Lapsley. Random early marking. In *QofIS '00: Proceedings of the First COST 263 International Workshop on Quality of Future Internet Services*, pages 43–54. Springer-Verlag, 2000.

[3] James Aweya, Michel Ouellette, and Delfin Y. Montuno. Weighted proportional window control of TCP traffic. *Int. J. Netw. Manag.*, 11(4):213–242, 2001.

[4] James Aweya, Michel Ouellette, Delfin Y. Montuno, and Zhonghui Yao. Enhancing network performance with TCP rate control. In *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, volume 3, pages 1712–1718, San Francisco, CA, 2000.

[5] Sally Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, October 1994.

[6] Sally Floyd. A report on some recent developments in TCP congestion control. *IEEE Communications Magazine*, 39(4):84–90, April 2001.

[7] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[8] Shrikrishna Karandikar, Shivkumar Kalyanaraman, Prasad Bagal, and Bob Packer. TCP rate control. *SIGCOMM Comput. Commun. Rev.*, 30(1):45–58, 2000.

[9] James F. Kurose and Keith W. Ross. *Computer Networks: A Top-Down Approach Featuring the Internet*. Addison Wesley, first edition, 2001.

[10] Jim Martin, Arne A. Nilsson, and Injong Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Trans. Netw.*, 11(3):356–369, 2003.

[11] netfilter. The netfilter/ip tables project. Available at http://netfilter.org/.

[12] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, second edition, 2000.

[13] VINT Group. UCB/LBNL/VINT network simulator (ns) - version 2. Available at http://www.isi.edu/-nsnam/ns.

[14] Cheng-Shong Wu, Ming-Hsien Hsu, and Kim-Joan Chen. Traffic shaping for TCP networks: TCP leaky bucket. In *Proc. of the IEEE Conference on Computers, Communications, Control and Power Engineering (TENCON)*, volume 2, pages 809–812, October 2002.